Monitoring and Controlling Content Access in KAD

Thibault Cholez, Isabelle Chrisment and Olivier Festor
MADYNES - INRIA Nancy-Grand Est, France
{thibault.cholez, isabelle.chrisment, olivier.festor}@loria.fr

***Abstract*—We propose a new distributed architecture that aims to investigate and control the spread of contents in the KAD P2P network through the indexation of keywords and files. Our solution can control the DHT at a local level with a new strategy bypassing the Sybil attack protections inserted in KAD. For the targeted DHT entries, we can monitor all requests emitted by the peers, from the initial content publication or search, to the final download request of fake files, assessing accurately peers interest to access it. We demonstrate the efficiency of our approach through experiments performed on the worldwide KAD network.**

***Index Terms*—P2P networks, DHT, Sybil attack, Honeypot, KAD**

I. Introduction

Peer-to-peer (P2P) networks are commonly used to share files within the Internet. Although peer-to-peer networks offer many advantages, they also provide support for harmful and malicious activities that can voluntarily propagate strongly undesirable content. As peer-to-peer systems are self-organized, dynamic and do not have a centralized infrastructure, measuring the spread of undesirable contents is a challenging task.

Our goal is to monitor and act on specific malicious contents given by law enforcement bodies (e.g. virus infected files, pedophilia content[1]...) and to observe the behavior of the related users in the widely deployed KAD P2P system. Active monitoring is an intrusive way of gathering information on a P2P system in that some traffic is injected in the network to gather information. Many crawlers have been used to study popular P2P networks like Gnutella [11] or e-Donkey [12]. Concerning KAD, a crawler [9] can discover the peers but can not monitor the shared contents. P2P honeypots [4] [5] are another way of monitoring contents in P2P networks. The common solutions consist in advertising fake files as normal peers in order to log the download queries received for these files, but without any guarantee that these files will attract all peers looking for the studied content.

In this paper, we propose a new approach, called HAMACK (Honeynet Architecture for Monitoring content ACcess in KAD), to monitor content access in a distributed hash table (DHT) and more specifically in a real P2P network called KAD. Our solution is based on distributed honeypots that passively monitor, with few resources ($\sim$20 nodes), all the requests issued by the peers for a specific content (keyword or file). To do so, we designed a new strategy to take control over some DHT entries that bypasses the protection mechanisms recently introduced in KAD to fight against Sybil attacks. Controlling the DHT enables many different functions from passive monitoring to the most achieved honeypot strategy including the eclipse of the indexed contents. Besides, by advertising attractive fake files which appear to be shared by many sources, HAMACK can assess peers' interest for a particular content from the initial content publication or search to the final download request of our files. These features allow our solution to surpass both traditional honeypots and active monitoring approaches to monitor and control content access.

This paper is structured as follows. Section II presents the related works. Section III describes how HAMACK can take control over KAD DHT entries. We describe, in Section IV, the functions that monitor or eclipse contents and promote honeypots with fake files. Section V deals with the implemented architecture of HAMACK and the results of experiments done in the real KAD network. Section VI concludes the paper and outlines our future works.

II. Related works

To overcome the limitations of a crawler [9] only focused on the peers, Steiner et al launched a Sybil attack on KAD [10]. The Sybil Attack, as described by Douceur [2], consists in creating a large number of fake peers called the "Sybils", and placing them in a strategic way in the DHT to take control over a part of it. Steiner et al injected 2^{16} Sybils from a single computer in a small zone of KAD. They were able to catch most of the Publish and Search requests within this zone. They achieved an eclipse attack making some keywords indexed on the Sybils disappear from the DHT.

Since then, the latest versions of the major KAD clients have introduced several protection mechanisms to mitigate this attack [1], in particular, it is no longer possible to infect a peer's routing table with Sybils having the same or very close IP addresses (same /24 subnetwork). Organizing a massive Sybil attack on KAD becomes impossible given the number of IP addresses involved. Even more localized attacks [10] are no longer possible as Sybils are directly injected in the routing table, which is now protected.

Some recent papers succeed to overcome these new constraints for their specific needs. Proposed recently in [8], *Montra* is a monitoring solution for KAD which is less intrusive than the Sybil attack. Like our approach, it relies on monitoring peers placed very close to the target to observe, in the case of *Montra*, they observe the traffic sent to other real peers. So, *Montra* is designed to conduct large-scale traffic studies but not to take control over specific contents.

[1]This work is partially funded by the French ANR Research Project MAPE (Measurement and Analysis of Peer-to-peer Exchanges for pedocriminality fighting and traffic profiling).

Considering security, [3] proposed an efficient way to achieve denial of service in KAD by delaying the *Search* function until a timeout reached. Their solution is focused on eclipsing content and they did not consider controlling content access.

To monitor and control content access in KAD, we adopt in this paper a more global strategy which can manipulate peers and investigate their behavior with honeypots. A subset of our strategy uses the index poisoning attack which was investigated in Overnet by [6]. The authors found out that the Overnet P2P network was subject to index poisoning attacks, performed to flood research results with fake or polluted files. Even if the global process is similar, the network (KAD vs Overnet) is different and we actually implemented the attack while [6] only investigated existing ones.

III. Exploiting the KAD DHT

A. Overview of KAD

KAD is a widely deployed (∼3 million concurrent online users) P2P network based on the Kademlia distributed hash table routing protocol [7]. It is implemented by the eMule and aMule clients, which allow users to share files.

Each node of KAD has a 128bits "KADID" setting its position in the DHT. All routing tasks are based on the XOR metric used to evaluate the distance between two peers, or between a peer and a content. Routing is done in an iterative way and with parallel lookups by using *KADEMLIA_REQ* requests to discover new peers close to a specific address. The routing table is composed of groups of contacts in a binary tree so that the closer an ID is to the current node, the better its knowledge of this part of the DHT.

As a file sharing application, the purpose of the KAD DHT is to index files and keywords. When a new file is shared, the raw data and all the keywords composing its name are hashed separately with a MD5 function generating a KADID for each piece of information. Those KADIDs are then published in the DHT. The peers able to index a file or a keyword are those that are close enough to the published hash. This distance is called the "tolerance zone" of a KADID, and is set to the first common 8bits (the most significant). The double indexing allows a peer to retrieve a file, given a set of keywords. To publish a file, two types of requests are sent:

- the *KADEMLIA2_PUBLISH_KEY_REQ* requests: sent towards the hash of the keyword to link it to a file
- the *KADEMLIA2_PUBLISH_SOURCE_REQ* requests: sent towards the hash of the file to link it to a source (a peer sharing the file)

After accepting a publication request for a given resource, a peer is in charge of indexing this specific content, and to answer the related search requests.

B. Controlling KAD indexation mechanisms

Localized attacks can still be effective in KAD with few distributed resources. The main weakness we exploit is the possibility left to the peers to freely choose their KADID. Our assumption is that if few numbers of modified clients choose their KADIDs very close to a given DHT entry, and considering KAD routing algorithm "return the X closest nodes...", the requests to this entry will arrive at these nodes with a very high probability. So, we focused our study on controlling KAD services through the routing process but without abnormally altering routing tables.

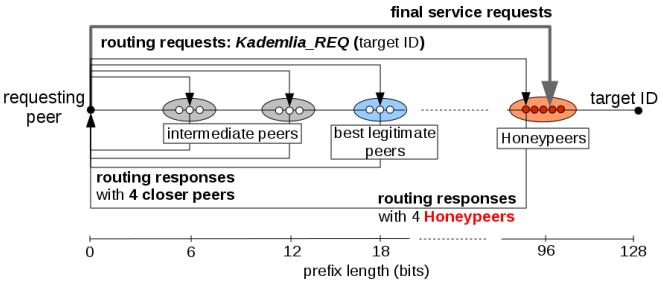

Fig. 1: Message exchange in the *Search* procedure

1) Positioning the Honeypeers: The first step taken to settle our honeynet is to position our peers accurately according to the keyword or the file over which we want to take control, i.e. closer than any other peer randomly choosing its KADID. As our modified clients are distributed and autonomous, we prefer to call them "Honeypeers" rather than Sybils. To compute the target ID and get the 128bits address, we need to apply the same MD5 hash function as the one used in a KAD client to the keywords or files we want to target. When this KADID is known, it is given as a parameter to each Honeypeer which can then derivate its own KADID from it, copying the first 96bits and choosing the remaining 32bits randomly.

Let N be the mean number of peers closer to the target than one of the Honeypeers. N is given by the probability for a peer to randomly choose a KADID with the same 96bits as the Honeypeers multiplied by the number of peers in the network. According to the latest estimations, the number of peers participating in KAD remains less than 5 million. The result in (1) shows that no legitimate peer will be placed closer to the target than our Honeypeers if they share 96 bits with it.

$$N = \frac{2^{32}}{2^{128}} \times 5 \times 10^6 = 6.31 \times 10^{-23} \quad (1)$$

2) Attracting requests during the "Search" procedure: As our Honeypeers are the closest to the targeted content, the critical point is to be sure that they can attract all the requests. All the services in KAD involving lookups in the DHT are achieved thanks to the "*Search*" object. Every keyword or file to be published, or every search of files or sources requested by a user, will generate an autonomous "*Search*" object in charge of all the requests management throughout the service process. Several *Search* objects can be managed by a client at the same time for different purposes, but all of them are distinct and autonomous.

The *Search* process has two separate parts. The first part consists in finding online peers in the tolerance zone of the target to fill the array of "*possible*" contacts, containing all

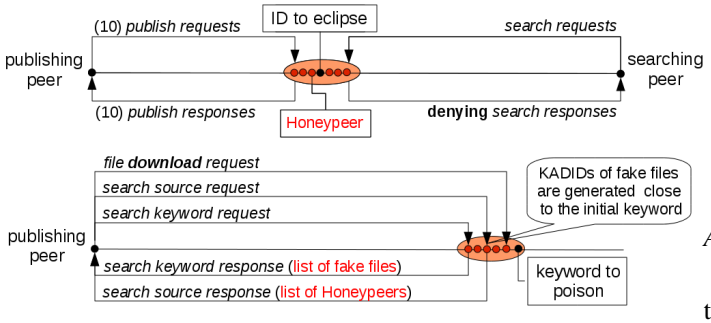

Fig. 2: HAMACK eclipsing content and poisoning a keyword

the suitable contacts found and sorted by distance. Several *KADEMLIA_REQ* are sent in parallel with the target ID as parameter. Routing is done in an iterative way with parallel lookups, as described in figure 1: first asking the closest 3 contacts from the routing table for even closer nodes toward the target ID, waiting for their responses and then reiterating by asking the 3 closest of the newly received contacts. Only the contacts that have successfully answered a *KADEMLIA_REQ* are considered available and can be used for the next step. Moreover, in order to avoid Sybil attacks, only one contact per /24 subnetwork can be selected. While closer contacts can be found in the last responses, the *Search* keeps looking for even closer contacts.

When the *Search* has not received any closer contact within the last 3 seconds, the second step is started. This phase consists in sending specific service requests (typically publish or search for keywords or sources) to the closest of the "*possible*" peers discovered in the previous step. According to the service set for the *Search* object, the request is different (for example *KADEMLIA2_PUBLISH_KEY_REQ* to publish a keyword). Finally, the stop conditions of the *Search* process are the following: either it gets more positive responses for the service than the replication rate, or it triggers a timeout. To protect the indexed entries from peers departure, a publication is considered successful when at least 10 peers have acknowledged it.

When a *KADEMLIA_REQ* arrives on one of our peers for the targeted hash, it only answers with other Honeypeers activated for this target ID. As the list of contacts managed by the *Search* object is always sorted by distance, all the Honeypeers found are placed in the best positions to receive the next requests. Therefore, thanks to this collaboration, as soon as peer looking for the target ID gets one of the Honeypeers among the routing responses, all the remaining service requests emitted by this peer will be sent to our peers.

IV. HAMACK FUNCTIONS

Controlling the indexing mechanisms allows HAMACK to provide different functions. They are described in the following subsections and in figure 2.

- Passive monitoring: monitoring all the publish and search requests for keywords and files;
- Contents eclipsing : removing keywords and files from the search engine;
- Index poisoning: replacing files linked to a keyword with fake ones appearing with a high number of sources;
- Promoting honeypots: replacing the peers sharing a file with our honeypots.

A. Passive monitoring and eclipsing contents

The simplest behavior consists in transparently monitoring the network by logging all requests concerning the target ID passing through the Honeypeers. Then, the Honeypeers answer each request like a normal client, without disturbing the network. When a *Publish keyword* request is received, HAMACK stores the following information from the request message: the sender IP address and port, the keywordID, the list of fileIDs and for each fileID the list of tags containing the file properties (full name, size). A *Publish source* request contains fewer fields: the fileID, the KADID, IP address and port of the source. This function allows HAMACK to monitor the activity of the targeted files (the peers sharing them) and keywords (the new published files). We can also monitor which peers are looking for a given content through the capture of *Search keyword* and *Search source* requests.

The second behavior aims to eclipse a target ID from the network as described in figure 2. As HAMACK manages to capture all the publish requests for the target ID, the Honeypeers only have to deny all the related incoming search requests to remove this entry of the DHT. This function is interesting because it allows HAMACK to prevent users from accessing to specific contents (i.e. files or keywords known to be malicious) by removing them from the search engine.

B. Index poisoning and final honeypots

When designing HAMACK, our main objective was to make our honeypot architecture for KAD as much efficient as possible. A naive honeypot architecture relies on several clients announcing fake contents. However, such a solution can not expect high efficiency without a huge amount of resources. If we rely on existing files shared by many peers, the honeypot can neither attract all requests for theses files, nor control the keywords through which they are referenced by the other peers. Therefore, it could receive many false positives (malicious files can be referenced with normal names). The use of generated fake solves this drawback but raises the problem of their visibility when compared to files that are already widely shared for the same type of content. To be attractive, the files have to appear with a high number of sources (peers sharing that file) in the search results. From a user point of view, the number of sources is the most important criteria to sort the search results: a high number means that the file is very popular, more reliable and can be downloaded fast. Our solution (figure 2) consists in taking control over the indexing scheme of KAD to control all the information returned about our fake files, including the estimated number of sources.

The first step is equivalent to eclipsing a keyword by attracting all the publish requests for the targeted keyword.

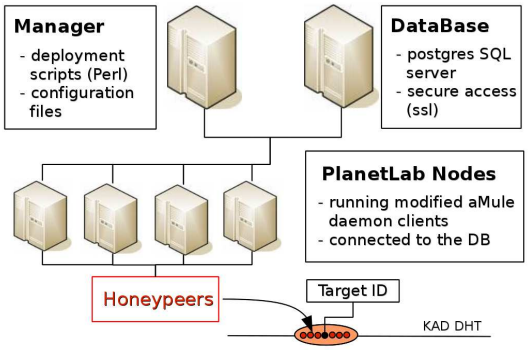

Fig. 3: Network architecture of HAMACK

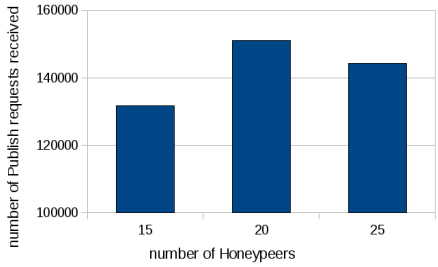

Fig. 4: Impact of the number of Honeypeers on HAMACK efficiency

Then, instead of denying the *Search Keyword* requests, all the Honeypeers composing HAMACK will answer with a list of fake files when requested. Because HAMACK has total control over the responses, each fake file will appear to be very attractive, with a high number of sources. When responding, we can choose whether the list is fully or partially composed of fake files.

Finally, when a fake file is selected, HAMACK receives the *Search Source* requests sent by the peer. To easily capture the *Search Source* requests emitted for all the fake files, each fileID is generated to be extremely close (96bits in common) to the associated keyword. With this optimization, every *Search Source* is attracted in the same way the initial search of keyword was, the Honeypeers remaining in the same place in the DHT. When sending the *Search Source* responses, HAMACK returns some Honeypeers (IP, port) to attract the final Download request of users.

V. Experiments on KAD

A. Implementation

The main constraint to be addressed by HAMACK is the IP address limitation inserted in the lastest version of the major KAD clients. To be effective, all the Honeypeers targeting a given ID must have different IP addresses and no more than 2 Honeypeers can belong to the same subnet. To fit with these constraints, HAMACK is run over few nodes of PlanetLab Europe, as described by figure 3. The Honeypeers execute a modified aMule client in daemon mode (aMuled). HAMACK can target several contents at the same time. Each node can easily handle dozens of clients on different ports and use each instance to target a different content. As long as the targeted entries are spaced on the DHT, the peers executed from a same computer should not enter in conflict in other peers' routing table and the IP address limitation will not reduce their visibility on the DHT.

The second major component of HAMACK is the database. It is implemented in a PostgreSQL server and is secured by an additional SSL module to avoid unauthorized connections. The database contains the parameters to configure an execution of HAMACK. In particular, the KADID of the targeted contents, the behavior to adopt for each of them (passive monitoring, eclipsing, poisoning), the list of fake files and the list of all the Honeypeers (KADID,IP and port) in charge of the same target ID to enable their collaboration. The database also stores all the requests related to the activity of the targeted contents that are monitored by our architecture. We designed the different tables to easily anonymize the IP address of the peers.

B. Measurements

To develop and evaluate HAMACK, we made several experiments on the real KAD network. As we absolutely did not want to disturb the network, we chose the keyword "document" as our test target ID because it is quite well indexed but insignificant when doing a real search on KAD.

1) Setting the parameters: The first experiments on KAD were run with the purpose to define the best values of the parameters affecting HAMACK's efficiency. The number of Honeypeers is an important parameter to consider to efficiently use our resources. The minimum number of Honeypeers to involve should be 10 (the replication factor to publish a content). But usually, more than 10 service requests can be sent and all of them have to be captured. The minimum number of Honeypeers used to start our experiments was 15. Graph 4 shows the evolution of HAMACK efficiency when the number of Honeypeers varies. The results show that using 20 Honeypeers instead of 15 improves the efficiency. However, the experiment with 25 Honeypeers shows an insignificant decrease of performance, simply meaning that there is no need to use more that 20 peers.

Many other parameters were studied. Time has no impact on HAMACK's efficiency but the number of received requests per hour varies within a day. The collaboration among Honeypeers (when asked for closer contacts with *Kademlia_REQ*) has a positive impact on the number of captured requests (+23%), but the number of contacts returned in a response is checked in recent clients. Answering with more than 4 contacts leads to a drop of the packet on these clients and a major decrease of HAMACK performances (-31%). Finally, actively advertising the Honeypeers in the network did not show any improvement in their attractivity.

2) HAMACK efficiency: We designed two experiments to evaluate the efficiency of our architecture to attract all the emitted requests. The first experiment consisted in publishing a

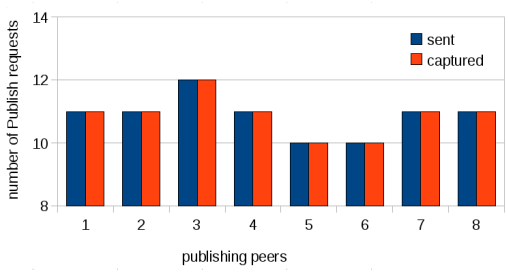

Fig. 5: Number of Publish requests sent from different publishing peers and captured

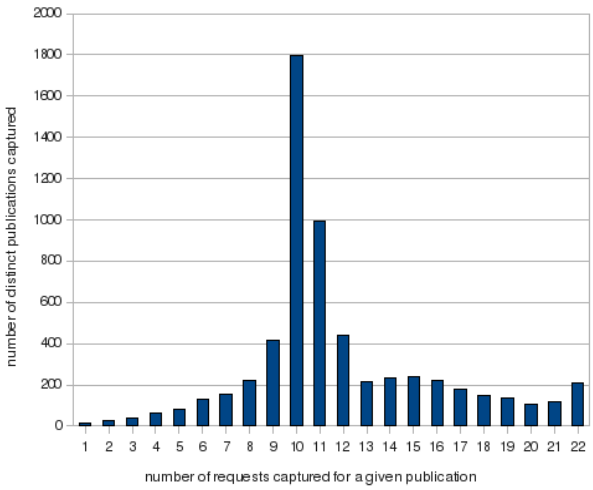

Fig. 6: Number of publications seen for a given replication

keyword managed by HAMACK from 8 different sources that we monitored. Each source had a different place in the DHT and was bootstraped from a different set of contacts, to avoid any mislead measurements due to improper initial conditions. We then can compare, for every publication, the number of replicated requests captured by HAMACK. Figure 5 shows that all the publish requests sent by our monitored clients are captured by the Honeypeers. Moreover, these very good results were confirmed by several search attempts performed on the target keyword ”document” when eclipsed by HAMACK: all the search requests were captured by the Honeypeers, and the global search ended without finding any result.

The second experiment considered how many publish requests are seen by HAMACK from the same peer in a short period of time. We know that the publication process needs 10 positive responses before stopping. Given that, each time that HAMACK sees less than 10 times the same request from a peer during the short time frame of a publication process, it means that the other replicated requests have been received by normal peers. The results displayed in figure 6 are very encouraging. Very few distinct publications are seen with less than 8 requests (509/5669). As expected, a very high number of distinct publications are seen around 10.

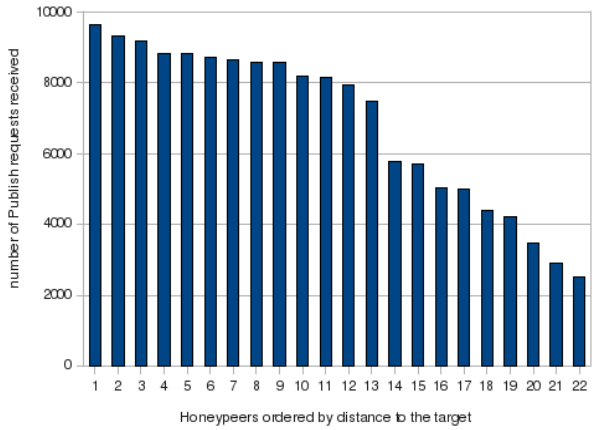

Fig. 7: Distribution of the load on the Honeypeers

We also evaluated how well the monitored requests are distributed on the Honeypeers. To do so, we counted the number of publish requests captured by each Honeypeer a day. Figure 7 shows that the load distribution between Honeypeers is related to the distance between each Honeypeer and the target ID: the closer a Honeypeer is to the target the more requests it receives. Even if the Honeypeers are closer to the target ID (96bits in common) than any other peer, and despite their collaboration to promote each other, they still compete with each other to receive all the replicated requests. The freedom to choose their KADID in the remaining 32bits is sufficient to account for a difference of their efficiency because of KAD routing algorithm. We see in the results that the Honeypeers with IDs (1-13) chose the same 97th bit as the target, while the Honeypeers (14-22) chose the other value, resulting in a decrease of their efficiency compared to the others. These results also indirectly show why normal peers, which are very far away from the target ID compared to the Honeypeers, have a very low probability to receive requests.

Finally, we evaluated the popularity of the fake files announced by HAMACK. In this last experiment, we poisoned the popular keyword ’spiderman’ with 4 files, 2 well shared and 2 with a low number of sources, while eclipsing the other references for this keyword a day. The search results returned by a client during our test are displayed in figure 8a. Our attack succeeds in eclipsing all the real entries and replacing them with our 4 fake files. Figure 8b shows, for each of these fake files, the proportion of ”first download request” received. The results show that the two well shared files are chosen first by 96% of users, which confirms that the the number of sources is a crucial factor when users have to choose among files and, therefore, to make an attractive honeypot.

C. Possible countermeasures

The current implementation of the *Search* process allows our Honeypeers to be very close to the target (i.e. 96 bits). A simple idea to mitigate the architecture is to discard the peers browsed during the *Search* which share a very high prefix with

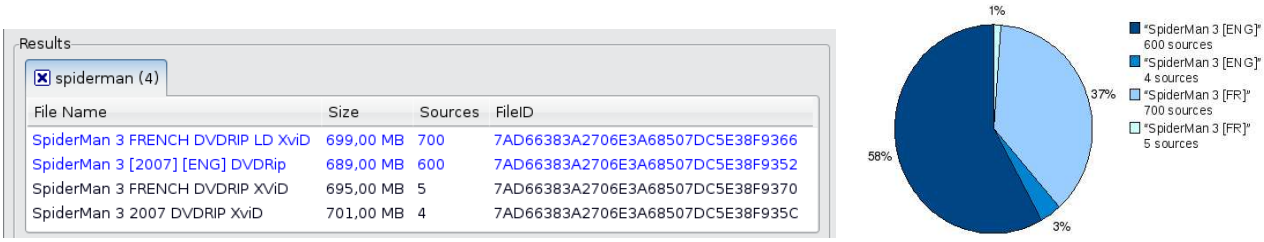

Fig. 8: Proportion of "first download requests" for the 4 fake files poisoning "spiderman"

the target. However, such a constraint can be easily bypassed by updating the way Honeypeers choose their KADID. In fact, even if a maximum prefix length is defined (for example 28 bits), there is always a window where the Honeypeers can be inserted, between the closest online peer to the target and the detection threshold. Moreover, introducing a restrictive constraint on the prefix will reduce the DHT efficiency by disabling close peers while a relaxed one would be ineffective.

A more likely way to detect our activity is to consider both prefix lengths and number of peers per prefix browsed during the *Search*. An accurate metric based on the distribution of peers around a target, regarding their prefix, could detect the few inserted Honeypeers from close legitimate ones. However, designing such a countermeasure which can efficiently mitigate our architecture seems very challenging. KAD scheme (more generally, DHTs) is by design vulnerable to localized attacks and can not be changed easily without breaking compatibility between clients.

VI. Conclusion

We have described HAMACK, an efficient Honeynet that bypasses the most recent protections of KAD against the Sybil attack. HAMACK uses several Honeypeers set very close to target entries of the DHT and these Honeypeers can take control over them. Our appproach does not rely on the injection of Sybils and is non-intrusive for the network besides the targeted contents (keywords and files). To investigate the P2P network, HAMACK provides highly efficient passive monitoring of all the incoming requests for the targeted DHT entries but can also control all the related responses which allow us to partially alter or fully eclipse any content indexed in KAD. The most accomplished function is the possibility to announce many files for a given keyword with realistic and attractive attributes, in particular the number of sources. Through the announcement of fake files, HAMACK can capture and control all the requests of peers, from the search of a keyword to the final download request, accurately assessing the interest of users for this specific content.

To achieve these functions, HAMACK exploits the weakness of KAD which allows the peers to freely choose their KADID and also relies on the very efficient *Search* process of the KAD DHT. Our work highlights a new dilemma of KAD which has to choose between its routing efficiency and the safety of its indexed contents. HAMACK is implemented by a lightweight architecture over PlanetLab and is fully operational. The first experiments run on the real KAD network showed a low upper bound of needed Honeypeers and that their collaboration increases the overall efficiency. Several experiments showed that HAMACK is very efficient in attracting the requests for the target IDs, which provides us full control over the indexed contents. Our final experiment poisoning a popular keyword confirmed the importance of controlling the number of sources to make an efficient honeypot.

Our future work consists in using HAMACK to monitor different types of malicious contents spreading in KAD. While we will use our architecture for this specific purpose, we are aware of the fact that it could also be used to disturb the network by eclipsing or poisoning good keywords and files. To close the loop, we are working on a revocation mechanism to fight against malicious peers detected by our architecture.

References

[1] T. Cholez, I. Chrisment, and O. Festor. Evaluation of sybil attacks protection schemes in kad. In *AIMS 2009: 3rd International Conference on Autonomous Infrastructure, Management and Security*, Twente, The Netherlands, 2009. Springer-Verlag.

[2] J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[3] M. Kohnen, M. Leske, and E. P. Rathgeb. Conducting and optimizing eclipse attacks in the kad peer-to-peer network. In *NETWORKING '09*, pages 104–116, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] M. Latapy, O. Allali, and C. Magnien. Measurement of edonkey activity with distributed honeypots. In *HOTP2P 2009. Sixth International Workshop on Hot Topics in Peer-to-Peer Systems*, Rome, Italy, 2009.

[5] H. Lee and T. Nam. P2p honeypot to prevent illegal or harmful contents from spreading in p2p network. In *The 9th International Conference on Advanced Communication Technology*, pages 497–501, Feb. 2007.

[6] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in p2p file sharing systems. In *INFOCOM*. IEEE, 2006.

[7] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[8] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach. Large-scale monitoring of dht traffic. In *IPTPS '09: 8th International Workshop on Peer-to-Peer Systems*, Boston, MA, April 2009.

[9] M. Steiner, T. En Najjary, and E. W. Biersack. A global view of KAD. In *IMC 2007, Internet Measurement Conference, October 23-26, 2007, San Diego, USA*, Oct 2007.

[10] M. Steiner, T. En Najjary, and E.W. Biersack. Exploiting KAD: possible uses and misuses. *Computer communications review, Volume 37 N5, October 2007.*

[11] D. Stutzbach and R. Rejaie. Capturing accurate snapshots of the gnutella network. In *8th IEEE Global Internet Symposium*, March 2005.

[12] J. Yang, H. M., W. Song, J. Cui, and C. Zhou. Crawling the edonkey network. In *GCCW '06: Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops*, pages 133–136, Washington, DC, USA, 2006. IEEE Computer Society.